3. Xdp : The Xpress Data Path.

The key design goal for XDP is to introduce programmability in the network datapath. The aim is to provide the XDP hooks as close to the device as possible (before the OS has created sk_buff metadata) to maximize performace while supporting a common infrastructure across devices. To support XDP like this requires driver changes. For an example see drivers/net/ethernet/broadcom/bnxt/bnxt_xdp.c. A bpf net device op (ndo_bpf) is added. For bnxt it supports XDP_SETUP_PROG and XDP_QUERY_PROG actions; the former configures the device for XDP, reserving rings and setting the program as active. The latter returns the BPF program id. BPF-specific transmit and receive functions are provided and called by the real send/receive functions if needed.

3.1 BPF_PROG_TYPE_XDP

- What do I do with it? XDP allows access to packet data as early as possible, before packet metadata (struct sk_buff) has been assigned. Thus it is a useful place to do DDoS mitigation or load balancing since such activities can often avoid the expensive overhead of sk_buff allocation. XDP is all about supporting run-time programming of the kernel in via BPF hooks, but by working in concert with the kernel itself; i.e. not a kernel bypass mechanism. Actions supported include XDP_PASS (pass into network processing as usual), XDP_DROP (drop), XDP_TX (transmit) and XDP_REDIRECT. See include/uapi/linux/bpf.h for the “enum xdp_action”.

- How do I attach my program? Via netlink socket message. A netlink socket – socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE) – is created and bound, and then we send a netlink message of type NLA_F_NESTED | 43 ; this specifies XDP message. The message contains the BPF fd, the interface index (ifindex). See samples/bpf/bpf_load.c for an example.

- What context is provided? An xdp metadata pointer; struct xdp_md * . XDP metadata is deliberately lightweight; from include/uapi/linux/bpf.h:

```
 /* user accessible metadata for XDP packet hook
  * new fields must be added to the end of this structure
  */
 struct xdp md {
          u32 data;
        __u32 data_end;
 };
```

- When does it get run? “Real” XDP is implemented at the driver level, and transmit/receive ring resources are set aside for XDP usage. For cases where drivers do not support XDP, there is the option of using “generic” XDP, which is implemented in net/core/dev.c. The downside of this is we do not bypass skb allocation, it just allows us to use XDP for such devices also.